Introduction to MATLAB

CS534 Fall 2016
What you'll be learning today

- MATLAB basics (debugging, IDE)
- Operators
- Matrix indexing
- Image I/O
- Image display, plotting
- A lot of demos
- ...

Matrices
What is a matrix?

Terms: *row*, *column*, *element*, *dimension*
How are the dimensions arranged

MxNxP matrix

First dimension

Second dimension

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\]
Defining a matrix with literals

```plaintext
>> A = [1 2 3; 4 5 6]
```

```
A =

1  2  3
4  5  6
```

semicolon separates rows
Defining a equally spaced vector

```matlab
>> A = 1 : 5
A =
     1     2     3     4     5
>> A = 1 : 2 : 10
A =
     1     3     5     7     9
```

Colon creates regularly spaced vectors

Bonus: what if I have something impossible like
```
A = -1 : 2 :-5
```
Demo
Define matrix with built-in functions

- zeros(M,N)
- ones(M,N)
- true(M,N)
- false(M,N)
- rand(M,N)
  - Create matrices with all 0/1/true/false’s
  - M, N are number of rows and cols respectively
  - can have more dims
- linspace(start, end, number)
  - Create linearly spaced vector ranging from start to end (inclusive)
  - number specifies the length of the vector

Bonus: How do you get a matrix of all 5?
Demo
Matrix Operations
size()

>> A = [1 2 3; 4 5 6];
>> size(A, 1)
ans =
    2
>> size(A, 2)
ans =
    3

A asks for first dimension

A asks for second dimension
size() cont'd

>> A = [1 2 3; 4 5 6];
>> [height, width] = size(A)

height =
   2

width =
   3
Concatenation

- $M = [A, B; C, D]$

Dimension must match; mark the next row
Concatenation in higher dims

- `cat(A, B, n)`

Operand matrices → Dimension to work on

The length of dimensions other than `n` of `A` and `B` must match
Demo
Linear Algebraic Operations

- Addition (dimensions match exactly)
- Subtraction (dimensions match exactly)
- Matrix Multiplication (MxN-matrix * NxP-matrix)
- Matrix Power (must be square matrix)
- Transpose
- Left Matrix Division (Solves A*x=B)
- Right Matrix Division (Solves x*A=B)
How Operations Work

\[ A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 1 \\ 5 & 6 \end{pmatrix} \]

\[ A + B = \begin{pmatrix} 4 & 3 \\ 8 & 10 \end{pmatrix} \quad A - B = \begin{pmatrix} -2 & 1 \\ -2 & -2 \end{pmatrix} \]

\[ A \times B = \begin{pmatrix} 13 & 13 \\ 29 & 27 \end{pmatrix} \quad A^{\times 2} = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix} \]

\[ A \backslash B = \begin{pmatrix} -1 & 4 \\ 2 & 1.5 \end{pmatrix} \text{ solves } A \times x = B \quad B \div A = \begin{pmatrix} -0.3077 & 0.3846 \\ 0.1538 & 0.6923 \end{pmatrix} \text{ solves } x \times A = B \]
Transpose

\[ C = \begin{pmatrix}
1 & 3 \\
5 & 7 \\
9 & 11 \\
13 & 15
\end{pmatrix} \]

\[ C' = \begin{pmatrix}
1 & 5 & 9 & 13 \\
3 & 7 & 11 & 15
\end{pmatrix} \]
Elementwise Operations

- dimensions need to match exactly
- usually use . to distinguish from their linear-algebraic counterparts

- +  Addition
- -  Subtraction
- .*  Element by Element Multiplication
- ./  Element by Element Division
- .^  Element by Element Power
  ○ A.^2  vs. A^2  vs. A.^B
Element-wise operations

\[ A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & 1 \\ 5 & 6 \end{bmatrix} \]

Note the 2 operand matrix for element-wise operations must match

\[ A ./ B = \begin{bmatrix} .333 & 2 \\ .6 & .666 \end{bmatrix} \]

\[ A .* B = \begin{bmatrix} 3 & 2 \\ 15 & 24 \end{bmatrix} \]

\[ A ^{\wedge} B = \begin{bmatrix} 1 & 2 \\ 243 & 4096 \end{bmatrix} \quad A ^{\wedge} 2 = \begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix} \]
Demo
Logical operators

- `==` is equal to
- `< > <= >=` less/greater than
- `~` not
- `~=` not equal to
- `&` elementwise logical AND (for matrices)
- `|` elementwise OR (for matrices)
- `~` negation

To be distinguished from

- `&&` short-circuit AND (for logical expressions)
- `||` short-circuit OR (for logical expressions)
Two useful commands

- `all()`
- `any()`
  - both work along one dimension of the matrix
  - by default compare along first dimension
  - use an optional second parameter to specify the dimension to work on
    - help to shrink a logical matrix to a logical scalar
    - then you can use `||` or `&&`
Demo
Matrix Indexing
Accessing a single element

$$A(2, 3)$$
Element on 2nd row, 3rd column

Note: indexing starts from 1, not zero!
Block Indexing

\[ A([1,2],[1,3]) \]
Can use vectors to index block of elements

\[ A([2,2],[1,2,3]) \]
Duplicate second row

\[ A([1,2],[3,2,1]) \]
\[ A([1,2],3:-1:1) \]
Change col orders
Indexing entire row/col

: represent the entire of that dimension
A(2, :)
Returns 2nd row

A(:, [1, 3])
Returns 1st, 3rd column in a matrix
end operator

dend represent the last of that dimension

```
>> A = [1 2 3; 4 5 6];
>> A(:, end:-1:1)
ans =
   3     2     1
   6     5     4
```

Reverse the col orders
end operator

```matlab
>> A = [1 2 3; 4 5 6];
>> A(:, [1 end 1])
ans =
    1     3     1
    4     6     4
```

Returns concatenation of 1st, last and 1st column
Indexing rules apply to 3D matrices too

\[ A(2, 4, 3) \]
Element on 2nd row, 4th column of the 3rd channel
Logical Indexing

Dimension of A and B must match

B

\[
\begin{pmatrix}
1 & 0 & 1 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

A

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
\end{pmatrix}
\]

>> A(B)

ans =

```
1
3
6
```

Select the element in A where B is true and put them in a **column vector**
Linear indexing

Indexes an element in a matrix with a single number/vector, in **column-major order**.

\[
A = \begin{pmatrix}
11 & 45 & 23 \\
21 & 89 & 59 \\
\end{pmatrix}
\]

**Bonus:** What will \(A(\_\_\_\_\_\_)\) be?
Demo
Element Assignment

- You can also index a matrix to assign values to its elements
- With scalar RHS, it’s easy
  - $A(1:2:end, 2:2:end) = 0$
- With matrix RHS, a little trickier
  - $A(1:2:end, 2:2:end) = A(1:2:end, 2:2:end) \times 2$
  - The dimension of the RHS must match the indexed block
- With empty matrix RHS, it’s a deletion
  - $A(2,:) = []$
  - You can only index whole row/col and delete them
Demo
Statistic functions can operate on the whole matrix

- sum
- mean
- max
- min
- median
- std
- var
  - By default they return col-wise statistics
  - use a second param to indicate the dimension to operate on
Vectorization
Vectorized code tends to be faster than non-vectorized code

\[
A = \text{rand}(1000,1000); \\
B = \text{rand}(1000,1000); \\
\text{for } i = 1:\text{size}(A,1), \\
    \text{for } j = 1:\text{size}(A,2), \\
        C(i,j) = A(i,j) + B(i,j); \\
    \text{end} \\
\text{end}
\]

Using loop: Elapsed time is 1.125289 seconds.
Vectorized code tends to be faster than non-vectorized code

\[ C = A + B; \]

Elapsed time is 0.002346 seconds.
Question: What's the sum of all elements greater than 2 in A?

```
summation = 0;
[height, width] = size(A);
for j = 1 : height
    for i = 1 : width
        if A(j, i) > 2
            summation = summation + A(j, i);
        end
    end
end
end
```
Question: What's the sum of all elements greater than 2 in A?

\[
\text{summation} = \text{sum}( A (A > 2) ) ;
\]

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & 0 & 1 \\
1 & 1 & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
3 & 4 & 5 & 6
\end{pmatrix}^T
\]

\[
\begin{pmatrix}
18
\end{pmatrix}
\]
A closer look at $\text{sum}(\text{A} \,(\text{A}>2))$

- $\text{A} > 2$ returns logical matrix
  - $[0 \ 0 \ 1; \\
    1 \ 1 \ 1]$

- $\text{A}([0 \ 0 \ 1; \ 1 \ 1 \ 1])$ returns column vector
- $[3 \\
  4 \\
  5 \\
  6]$

- $\text{sum}([3 \ 4 \ 5 \ 6]')$
More Examples

Logical indexing on the right-hand side
\[
\text{count} = \text{sum}(2 < A \mid A == 5); \\
\text{ave} = \text{mean}(A(\text{mod}(A, 2) == 0));
\]

Logical indexing with assignment
\[
A(\text{isnan}(A)) = 0;
\]
Demo
Images
What is an image?

An RGB image is a 3D $M \times N \times 3$ matrix.
The "layers" in a color image are often called *channels*
filename = 'badgers.jpg';
im_orig = imread(filename);

% your image processing routines

figure; imshow(im_orig);
figure; imshow(im_processed);
imwrite(im_processed, 'hw.jpg');
Important image related functions

- **imread**  Read image from disk
- **imwrite** Write image to disk
- **figure**  Create new figure
- **imshow**  Display image
- **im2double** Convert image datatype to double
- **im2uint8** Convert image datatype to uint8
- **rgb2gray** Convert a 3-channel rgb image to a single channeled gray image
Be aware of your return types!

### Algorithms
For most image file formats, `imread` uses 8 or fewer bits per color plane to store image pixels. This table lists the class of the returned image array, A, for the bit depths used by the file formats.

<table>
<thead>
<tr>
<th>Bit Depth in File</th>
<th>Class of Array Returned by <code>imread</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit per pixel</td>
<td><code>logical</code></td>
</tr>
<tr>
<td>2 to 8 bits per color plane</td>
<td><code>uint8</code></td>
</tr>
<tr>
<td>9 to 16 bits per pixel</td>
<td><code>uint16 (BMP, JPEG, PNG, and TIFF)</code></td>
</tr>
</tbody>
</table>

For the 16-bit BMP packed format (5-6-5), MATLAB returns `uint8`.

### Output Arguments
- **B — Filtered image**
  - Numeric array the same size and class as input image
  - Filtered image, returned as an array of the same size and class as the input image.

---

`imread()`

`imfilter()`
Be aware of your argument types!

**interp2()**

**imhist()**
Common bugs

This wouldn’t work, but why?

```matlab
im_original = imread('lena_gray.jpg');
[Xq, Yq] = meshgrid(0.5:99.5, 0.5:99.5);
im_interp = interp2(im_original, Xq, Yq);
```

Pay close attention to:
- input/output types of the function
- range of the corresponding image types
`subplot()` squeezes more into a figure

`subplot(m, n, p);`

- `p` is based on an `m x n` grid.
- `p` determines location of this subplot; it's an index in row-major order.
Demo
Reference

Matlab presentation from previous semester

Matlab presentation this time:
  part1
  part2
  demo files

MATLAB cheat sheet

Note: you may need to login to your wisc.edu google drive to view the files from this time
fin.